Disk Cache

Reference to Related Applications

This application relates to U.S. Patent Application Serial No.: _____, entitled "Data Storage Device", filed on the same day as this application, and incorporated by reference herein.

Background

Computer storage devices vary widely in their characteristics. For example, memory chips can quickly respond to requests to store or retrieve data. Most chips that permit data storage and retrieval, however, require some amount of electricity to "remember" the stored data. Computer disks, on the other hand, generally remember stored data even without power. Additionally, a disk typically stores much more data than a chip. Unfortunately, due to their mechanical components, traditional disks are often appreciably slower than other storage mediums.

In many ways, a computer disk is much like an old fashioned record player. That is, most disks include a platter spun by a rotating spindle. A disk head, controlled by a disk head actuator, moves about the spinning platter reading or writing information in different platter locations. Thus, to read or write data, the disk head actuator moves the disk head to await the location of interest to spin underneath.

A variety of techniques have been devised to mask the comparatively slow speed of disks. For example, many systems feature a cache (pronounced "cash") formed from one or more memory chips (e.g., DRAM chips). The cache stores a changing subset of information also stored by a disk. When the cache

holds a requested block of data, the cache can quickly respond to a request without awaiting disk retrieval.

Summary

In general, in one aspect, the disclosure describes a cache that includes a front-end interface that receives data access requests that specify respective data storage addresses, a back-end interface that can retrieve data identified by the data storage addresses, cache storage formed by at least two disks, and a cache manager that services at least some of the requests received at the front-end interface using data stored in the cache storage.

Embodiments may include one or more of the following features. The front-end interface may conform to a protocol such as SCSI (Small Computer System Interface), Fibre Channel, Infiniband, and/or IDE (Integrated Device Electronics). The disks may platters less than 3.5 inches in diameter such as 2.5 inch, 1.8 inch, and/or 1 inch diameter platters. The cache may implement a RAID (Redundant Array of Independent Disks) scheme using the disks. The cache may request data from a back-end storage system, retrieve requested data from the disks, send data to the back-end system for writing, determine the location of back-end system data within the disks, and/or remove data from the disks.

The addresses may specify storage locations of a back-end storage system that includes a collection of one or more disks. The data storage addresses may be within an address space, for example, of back-end storage or of a different cache. The requests may be I/O (Input/Output) requests.

In general, in another aspect, the disclosure describes a

method of servicing data access requests at a cache having cache storage formed by at least two disks. The method includes receiving the data access requests that requests specify different respective data storage addresses and servicing at least some of the requests using data stored in the disks.

In general, in another aspect, the disclosure describes a data storage system that includes a back-end storage system having an address space where addresses in the address space identify blocks of storage. The system also includes a cache for the back-end storage system that has a lesser storage capacity than the back-end storage system. The cache includes a front-end interface that receives I/O (Input/Output) requests that specify respective addresses of back-end storage blocks, a back-end interface that communicates with the back-end storage system, cache storage formed by at least two disks having platter diameters less than 3.5 inches, and a cache manager that services at least some of the I/O requests received via the front-end interface using blocks temporarily stored in the cache storage.

Brief Description of the Drawings

FIG. 1 is a diagram illustrating a cache.

FIG. 2 is a flow-chart of a sample process implemented by a cache.

FIG. 3-7 are diagrams illustrating operation of a cache.

FIG. 8 is a diagram illustrating a series of caches.

Detailed Description

Disk capacities have grown enormously, doubling roughly every 18 months. Unfortunately, disk access rates have generally failed to keep pace with this growth, doubling only every 34 months. This growing gap between drive capacity and access rates

implies that although disk drives can hold more data, the data cannot be accessed as often. A cache can help mask the growing disparity between disk capacities and data access rates by temporarily storing "hot" (e.g., frequently accessed) disk data. Generally, the effectiveness of a cache depends on the size of the cache relative to the amount of data accessed through it. While increasing the size of a cache can help speed data access, construction of larger memory-based caches can be costly due to the expense of memory chips.

FIG. 1 illustrates a cache 100 that uses a collection of disks 110 for cache storage instead of memory. Since the slow speed of disks usually contributes to the need for a cache 100, a cache 100 featuring disks 110 may seem counter-intuitive. However, spreading cached data across a large number of disks 110 in the cache 100 can increase the number of disks 110 involved in responding to a set of requests. While the delay associated with the mechanics of accessing data from an individual disk 110 may remain unchanged, spreading cached data across the disks 110 increases the number of disks simultaneously seeking information and can increase overall transaction rates. In other words, while increasing the number of spindles and independently actuated read/write heads may not necessarily increase the servicing of a single request, amassing the spindles may increase the performance of a cache handling multiple requests.

A cache 100 using the collection of disks 110 can potentially offer a number of advantages. For example, a larger cache may be constructed more cheaply than a similarly sized cache constructed from memory. Additionally, the disks 110 can preserve data without power, potentially eliminating a need for backup power often featured in memory-based cache systems. Further, as described below, the disks 110 may feature better

power and thermal characteristics than a cache constructed from memory. These and other potential advantages are described below.

In greater detail, FIG. 1 illustrates an example of a system that uses a cache 100 to service requests for back-end storage 102. Back-end storage 102 may be constructed from a wide variety of devices such as disks and/or memory chips. For example, the back-end storage 102 may be a large storage subsystem formed by a collection of disks.

As shown, the cache 100 includes a front-end interface 104 that communicates with hosts (e.g., a remote computer), or other components. For example, the front-end interface 104 may receive an Input/Output (I/O) request for a block of back-end storage 102 specified by a data storage address. While not a requirement, the interface 104 may conform to a wide variety of different protocols such as SCSI (Small Computer System Interface), Fibre Channel, or Infiniband. Thus, the interface 104 can present a standard front-end offered by a variety of other devices. The interface 104 can hide details of cache 100 construction. That is, a host or other system component need not have any knowledge that the cache 100 uses disks 110 for cache storage instead of memory.

As shown, the cache 100 also includes a back-end interface 108 that communicates with back-end storage 102. For example, the back-end interface 108 can communicate with back-end storage 102 to request data not currently stored in the cache 100 disks 110. The back-end interface 108 may also conform to a wide variety of communication standards used by back-end storage systems 102 such as SCSI, Fibre Channel, and so forth.

The cache manager 106 implements cache logic. For example, the manager 106 can track the continually changing set of data

cached by the disks 110, determine if requests can be satisfied by the cache 100 disks 110, forward commands to the back-end interface 108 to store or retrieve back-end storage 102 data, instruct the front-end interface 104 how to respond to the request, and so forth. The caching operations described above are merely examples of caching features and not an exhaustive list. The cache 100 can be configured to use a vast number of other caching techniques.

To store and retrieve data from the cache 100 disks 110, the manager 106 may use a wide variety of techniques. For example, the manager 106 may implement a RAID (Redundant Array of Independent Disk) scheme to improve the performance of the disk 110 array and/or protect against media failure. Different RAID schemes can feature different combinations of techniques known as striping, mirroring, and error correction.

RAID schemes that use striping divide data being stored into different portions and store the portions on different disks 110. For example, striping of the data "EMC Corporation" may result in "EMC C" being stored in a first disk, "orpor" in a second disk, and "ation" in a third disk. To retrieve the data, all three disks can operate concurrently. For example, disks 1, 2, and 3 may all simultaneously seek their portion of the "EMC Corporation" data. Thus, a block of back-end storage data may be distributed across multiple disks.

RAID schemes that use mirroring store copies of data on different disks. For example, two different disks may store a copy of the data "EMC Corporation". While storing data requires writing the information to both disks, the data can be retrieved from either device. For example, if one device is busy, the other can be used to retrieve the data. Mirroring also provides

an increased measure of security. For example, if one disk malfunctions the other disk can be used.

Many RAID schemes also feature error correction/detection techniques. For example, many RAID schemes store an error correction code such as a Hamming Code for data being stored. The code can be used to reconstruct data even though some error occurred.

Alternatively, the manager 106 may use data storage techniques other than RAID. For example, the manager 106 may map different address ranges to different cache disks 100. Again, such a technique can increase the likelihood that different disks 110 will concurrently seek requested data.

A variety of disks 110 may be accessed by the cache 100. Each disk may feature its own spindle and independently operating disk head actuator. Alternatively, a portion of the disks 110 may share a common spindle and actuator. While the cache 100 may feature standard 3.5-inch diameter platter drives 110, the cache 100 may instead feature a large collection of ultra-dense, small capacity drives 110 having smaller platters. For example, the cache 100 may be assembled from 2½-inch or 1.8-inch diameter platter disks typically produced for laptop computers. The cache 100 may also be assembled from even smaller devices such as IBM's MicroDrive™ that features a 1-inch platter, roughly the size of a quarter.

Using disks 110 in the cache 100 can offer a number of potential benefits over conventional memory-based caches. For example, current memory technology (e.g., a 128 MB DIMM) dissipates approximately 26 mWatts per MB. In comparison, a typical currently available disk drive only dissipates approximately .63 mWatts per MB. Thus, using disks 110 instead of memory may substantially lessen the need for costly, high

performance cooling systems that can place practical limits on the size of memory-based caches. Additionally, due to the lower capacity cost and power dissipation of disks 110, it is possible to build much larger caches than what may be practical with memory based approaches. The lower power dissipation of the disks 110 can potentially permit larger caches to be packed into much smaller volumes than those based on memory technologies.

Though described in FIG. 1 as having independent components 104, 106, 108, an implementation may feature a monolithic design having a processor that executes instructions for the front-end interface 104, back-end interface 108, and cache manager 106. Additionally, the cache 100 may provide other features. For example, the back-end manager 108 may be configured to manage components (e.g., disks) of back-end storage 102.

The cache 100 may provide or be configured with disk interfaces (e.g., SCSI or IDE [Integrated Disk Electronics]) that allow a system manager to attach disks to the cache 100 as desired. For example, a system manager may add disks to increase the size of a cache. Co-pending U.S. Patent Applcation Serial No. _____, entitled "Data Storage Device", describes a sample implementation of a multi-disk device in greater detail.

FIG. 2 illustrates a sample caching process 130 that uses a collection of disks. For write requests, the process 130 can store 136 data in the cache disks for subsequent writing 140 to back-end storage.

For read requests, the cache determines 138 whether the requested data is already stored in the cache disks. If so, the cache can retrieve 142 the requested data without involvement of the back-end storage. If not, the cache can retrieve 144 the data from back-end storage and store 146 the retrieved data in the cache disks, for example, using a RAID or other storage

technique. The caching approach depicted in FIG. 2 is merely illustrative. That is, the cache may use a different algorithm to determine whether to cache data. For example, other algorithms may cache based on a history of requests instead of the most recent.

Again, the cache may perform a number of other caching tasks not shown in FIG. 2. For example, The cache 100 may periodically or on an as-needed basis remove cached data, for example, using an LRU (Least Recently Used) algorithm. Additionally, the cache may store tag data identifying the back-end storage addresses stored by the cache and the location of the corresponding blocks within the cache disks. This tag data may be stored on the cache disks or on some other storage device, for example, upon detecting a power failure. The tag data enables a data storage system to recover after a failure, for example, by identifying the cache location of deferred writes or other information in the cache.

FIGs. 3 to 7 show an example of cache 100 operation. As shown, the cache 100 services I/O requests for blocks of back-end storage 102. As shown, the back-end storage 102 features an address space 206 of data storage ranging from address "00000000" to "FFFFFFFF". One or more back-end storage 102 disks may provide the disk storage identified by this address space 206.

As shown in FIG. 3, the cache 100 receives an I/O request 200 for the back-end storage 102 block 202 specified by data storage address, "000000001". As shown in FIG. 4, after the cache manager 106 determines that the cache disks 110 do not currently store the requested block 202, the cache manager 106 instructs the back-end interface 108 to retrieve the block 202 from back-end storage 102. As shown in FIG. 5, the cache

manager 106 stores the retrieved block 202 in the cache disks 110. As shown, the cache manager 106 has divided storage of the block between disks 110a and 110b (shaded). The cache manager 106 also causes the front-end interface to respond to the request 200 with the retrieved block 202. When the cache 100 receives a request for the same block (shown in FIG. 6), the cache 100 can quickly respond to the request by retrieving the block from disks 110a and 110b, as shown in FIG. 7.

As shown in FIG. 8, the cache 100 may be one of a series of caches 210, 100, 212. For example, cache 210 may initially receive an I/O request for a block of data. If the cache 210 does not currently store the requested block, the cache 210 may send a request to cache 100. Similarly, if cache 100 does not currently store the requested block, the cache 100 may send a request to cache 212, and so forth, until a cache or back-end storage 102 provides the requested block.

A cache may feature its own address space. Generally, such address spaces will increase along the chain of caches 210, 100, 212. Potentially, caches may have knowledge of other cache's address spaces. Thus, a request between caches may feature an address of a cache's address space instead of a back-end storage address space.

The techniques described herein are not limited to a particular hardware or software configuration and may find applicability in a wide variety of computing or processing environments. The techniques may be implemented in hardware or software, or a combination of the two. Preferably, the techniques are implemented in computer programs executed by a cache processor and a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements).

Each program may be implemented in high level procedural or object oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case the language may be compiled or interpreted language. Each such computer program may be stored on a storage medium or device (e.g., ROM, CD-ROM, or hard disk) that is readable by a programmable processor.

Other embodiments are within the scope of the following claims.